Metodika Git Flow

Tím PARKety, č. 15

Vedúci projektu: Ing. Ivan Srba, PhD.

Predmet: Tímový projekt I

Ročník: 2017/2018

Vypracoval(i): Miroslav Lehotský

Babinec Peter, Bc. Hoang Martin, Bc.
Hučko Jakub, Bc. Karas Marek, Bc.
Lehotský Miroslav, Bc. Mičo Jakub, Bc.
Púčať Samuel, Bc. Vnenčák Stanislav, Bc.

21. novembra 2017

1 Úvod

Pre kolaboráciu na projekte sme sa rozhodli používať verzovací systém git. Git je open source distribuovaný verzovací systém, čo znamená, že každá lokálna kópia centrálneho repozitára obsahuje celú históriu verzií, narozdiel od centrálnych verzovacích systémov ako napr SVN.

2 Centrálny repozitár

Pre umiestnenie nášho centrálneho git repozitára sme sa rozhodli použiť GitHub. Repozitár na GitHube je vytvorený ako privátny u jedného z členov tímu, to znamená, že repozitár nieje prístupný verejnosti ale prístup k nemu je umožnený len na základe explicitnej pozvánky od vlastníka repozitára.

3 Prerekvizity

Pre prácu nad spoločným repozitárom je nutné mať nasledujúce veci:

- Mať užívateľský účet na github.com
- Byť členom privátneho repozitára ako kolaborátor
- Mať na svojom počítači nainštalovaný program git

4 Práca s gitom

4.1 Naklonovanie centrálneho repozitára

Prvým základným krokom, ktorý je nutné vykonať ak chce niekto pracovať na projekte, je naklonovanie repozitára. Klon repozitára je možné dostať dvomi spôsobmi:

- Protokolom https - nutno pri spojení zadať prihlasovacie údaje do githubu

- Protokolom ssh - nutno mať vygenerovaný pár ssh kľúčov: verejný a súkromný, verejný kľúč musí byť nahraný na github do užívateľského profilu

V oboch prípadoch sa použije ten istý git príkaz pre vytvorenie lokálnej kópie centrálneho repozitára so špecifickou url podľa použitého protokolu:

```
$ git clone <https_url/ssh_url>
```

4.2 Stiahnute zmien z centrálneho repozitára

Predtým, ako začneme vytvárať novú branch, alebo v rámci nejakej branche vykonávať zmeny, mali by sme sa vždy presvedčiť, že sme na najnovšej verzii s remote/develop, resp. remote/<nazov_branche>. Ak sme up-to-date, netreba sťahovať žiadne zmeny. Ak výpis príkazu git status dá niečo ako:

```
On branch <nazov_branche>
Your branch is behind '<remote>/<nazov_branche>' by 1 »
  » commit, and can be fast-forwarded.
```

Stiahnutie zmien do lokálneho repozitára vykonáme nasledovne:

```
$ git pull <remote> <nazov_branche>
```

Ak sme v lokálnom repozitári mali necommitnuté zmeny, v nejakom súbore, v ktorom boli vykonané zmeny aj v commitoch, ktoré sa snažíme stiahnuť z centrálneho repozitára, dostaneme konflikt:

```
error: Your local changes to the following files would be »
  » overwritten by merge: [list_konfliktnych_suborov]
Please, commit your changes or stash them before you can »
  » merge.
Aborting
```

Jedným riešením je teda commitnúť tieto zmeny, následne bude treba urobiť merge, v prípade nekonfliktných zmien v rámci súboru (zmeny sa neprekrývajú, napríklad zmeny na rôznych riadkoch v súbore), vyrieši to auto-merge. V prípade konfliktných zmien bude treba vykonať merge ručne otvorením príslušného súboru a vykonaním zmien.

4.3 Vytvorenie novej branche

Pred začatím implementácie nejakej story treba vždy vytvoriť novú branch z develop branche. Keďže chceme byť čo možno najviac up-to-date, najprv sa presvedčíme, že máme u seba v lokálnom repozitári najnovšiu serverovú verziu develop branche. To vykonáme stiahnutím zmien z centrálneho repozitára.

Následne v našom lokálnom repozitári vytvoríme novú branch. To sa docieli nasledovným príkazom vo vnútri lokálneho git repozitára, tento príkaz nás rovno aj prehodí na novo vytvorenú branch:

```
$ git checkout -b <nazov_branche>
```

Po vykonaní príkazu by mal git vypísať do konzoly:

```
Switched to a new branch '<nazov_branche>'
```

Pre výpis dostupných branchi v lokálnom repozitári a tiež pre overenie na akej branchi sa práve nachádzame možno zistiť príkazom:

```
$ git branch
```

4.4 Commitovanie zmien

Po dokončení implementácie nejakej samostatnej časti funkcionality, by mali byť tieto zmeny commitnuté. Pre zobrazenie zmenených súborov od posled-

ného commitu možno použiť príkaz:

```
$ git status
```

Najprv by tieto zmeny mali byť trackované v sekcii "Changes not staged for commit". Potom treba vybrať zmeny, ktoré chceme commitnúť, to sa vykonáva nasledovne:

```
$ git add <nazov_suboru>
```

Prípadne pridaním automaticky všetkých zmien:

```
$ git add [--all|-A]
```

Následne treba skontrolovať, zase pomocou príkazu **git status**, ktoré zmeny sa nachádzáju v sekcii "Changes to be committed", všetky tieto zmeny budú po commitnutí nahrané, preto ak sa tam priplietla dáka zmena alebo súbor, ktorý nechceme trackovať a mať v repozitári, možno ho z tohto stagingu odobrať príkazom:

```
$ git reset HEAD <nazov_suboru>
```

Potom, ako máme skontrolované a sme si istý, že chceme zmeny commitnúť, urobíme to príkazom:

```
$ git commit -m "<text_commitu>"
```

Ako text commitu je vhodné dávať výstižný opis vykonaných zmien, prípadne aj s číslom tasku/story z JIRA, napr: *'Integracia so senzormi #TP-18'*
Po commitnutí zmien sú zmeny nahrané zatiaľ len v lokálnom repozitári, po príkaze git status dostaneme:

```
On branch <nazov_branche>
Your branch is ahead of '<remote>/<nazov_branche>' by 1 »
  » commit.
```

Históriu všetkých commitov si vieme pozrieť príkazom:

```
$ git log
```

4.4.1 Úprava posledného commitu

Je celkom bežná záležitosť, že pri pridávaní súborov do stagingu sme zabudli jeden súbor pridať, alebo sme napr. zabudli v jednom z týchto súborov odstrániť nežiaduci výpis do konzoly a takéto zmeny sme commitli. Aby sme v takom prípade nemuseli vytvárať celkom nový commit, jednoducho ten posledný len uprávime, aby obsahoval aj zabudnuté zmeny.

Takže odstránime v danom súbore napr. nežiaduci výpis do konzoly, pridáme súbor do stagingu a vykonáme:

```
$ git commit --amend
```

git nás vtedy vyzve na zadanie nového message commitu, ak by sme ho nechceli meniť a nechať pôvodný, možme použiť ten istý príkaz s dodatočým argumentom:

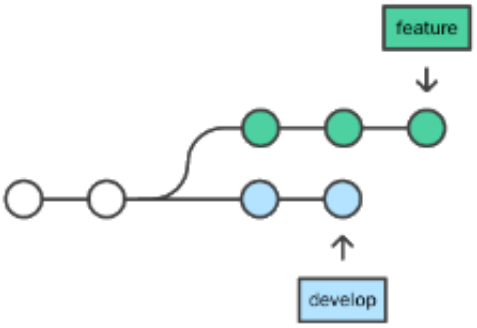
```
$ git commit --amend --no-edit
```

Rovnako vieme urobiť opačnú vec a to ponechať všetky zmeny commitu rovnaké ale zmeniť jeho message:

```
$ git commit --amend -m "New message"
```

4.5 Mergovanie feature branche s develop branchou

Ak chceme v našej lokálnej feature branchi najnovšie zmeny z develop branche, pričom vo feature branchi už máme nejaké vlastné commity a develop takisto, teda tieto dve brache sa rozchádzajú:



Obr. 1: *Branch develop a feature sa rozchádzajú*

Ako prvé musíme stiahnuť zmeny z centrálneho repozitára (monžno vynechať, ak mergujeme s lokálnou branchou):

```
$ git fetch <remote>
```

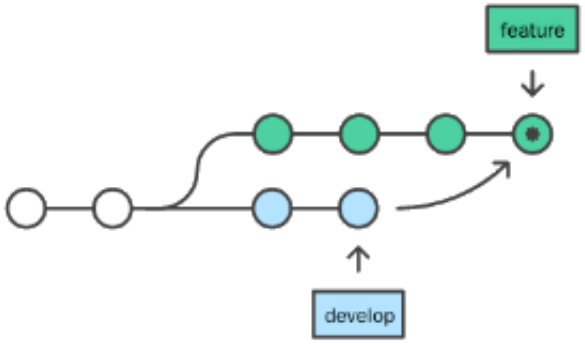
Tento príkaz nám stiahne všetky branche z centrálneho repozitára, ktoré uloží do: <remote/nazov_branche>, teda napr pre remote = origin a branch develop => nazov_branche = origin/develop.
Pre ich spojenie máme dve možnosti:

4.5.1 Merge

Spojenie dvoch branchi pomocou merge vykonáme nasledovne:

```
$ git merge <nazov_branche>
```

Tento spôsob mergu vytvorí pre merge nový plnohodnotný commit, ktorý môžme vidieť v histórii.



Obr. 2: *Merge branch develop do feature*

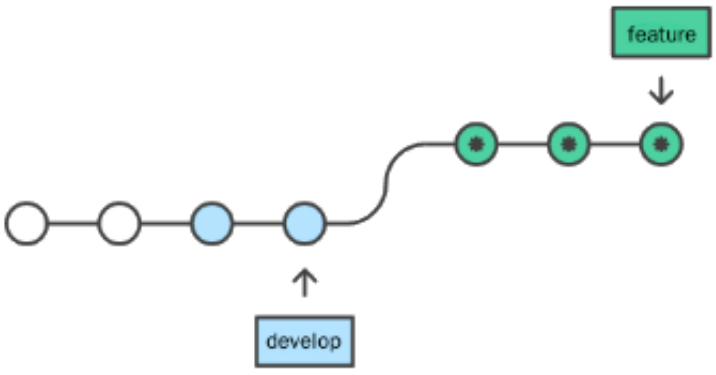
Ak akceptujeme zmeny develop branche do našej feature často, je lepšie použiť rebase, pretože tak nevytvoríme veľa merge commitov a história bude prehľadnejšia.

4.5.2 Rebase

Spojenie dvoch branchi pomocou rebase vykonáme nasledovne:

```
$ git rebase <nazov_branche>
```

Tento spôsob nevytvorí žiadny nový commit, ale upraví všetky commity vo feature branchi, od posledného zhodného commitu s danou <nazov_branche>



Obr. 3: *Rebase branch feature nad develop*

4.6 Pushovanie zmien do centrálneho repozitára

Aby sme lokálne zmeny nahrali aj do centrálneho repozitára na serveri, vykonáme nasledovný príkaz:

```
$ git push <remote> <nazov_branche>
```

<remote>: vzdialený repozitár na serveri, pri klonovaní git automaticky vytvoril remote 'origin' asociovaný s URL, z ktorej sme repozitár klonovali. Remote repozitáre vieme pridávať, meniť, odoberať, pre výpis všetkých remote repozitárov asociovaných s lokálnym git repozitárom vypíšeme pomocou:

```
$ git remote -v
```

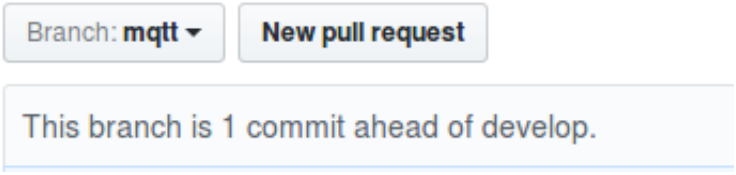
Prípadne si vieme pozreť git config aj s ďalšími nastaveniami repozitára, config súbor sa nachádza uprostred zložky repozitára a v jeho skrytej zložke .git: **.git/config**.

Po pushnutí zmien by sa mali zmeny nahrať do príslušného remote centrálneho repozitára a príslušnej branche, ktoré sme určili. Výpis príkazu git status by potom mal vyzerať nasledovne:

```
On branch '<nazov_branche>'
Your branch is up-to-date with '<remote>/<nazov_branche>'
nothing to commit, working directory clean
```

4.7 Nahranie zmien do develop branche

Za aktuálny stav celého projektu sa považuje stav projektu v branchi **develop**, ktorá je chránená. Kód do tejto branche nie je možné priamo pushovať. Keď už máme funkcionalitu/story/task/bug fix hotový, chceme ho dostať do develop branche. To vykonáme vytvorením **pull request**u na GitHub repozitáry, je možné tam pridať reviewera, pridávať komenty, ...



Obr. 4: *Na príslušnej branchi vytvoriť nový pull request*

Obr. 5: *Vytváranie pull requestu, branch vs branch*

4.8 Zmazanie feature branche

Ak bola branch vytvorená len pre vývoj konkrétnej funkcionality, po dokončení vývoja a úspešnom nahraní zmien do develop branche ju treba zmazať.

Zmazanie branche v lokálnom repozitári vykonáme:

```
$ git branch -d <nazov_branche>
```

Zmazanie na githube docielime nasledovným príkazom:

```
$ git push <remote> --delete <nazov_branche>
```

Pre všetky možné prepínače k jednotlivým git príkazom si možno vždy pozrieť manuál s ich popismi pomocou:

```
$ man git <prikaz>
```